Summary Report: A Preliminary Investigation into the Use of Fuzzy Logic for the Control of Redundant Manipulators

John B. Cheatham, Jr.
Kevin N. Magee

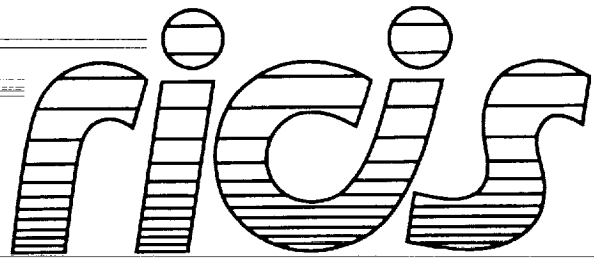
Department of Mechanical Engineering and Materials Science
Rice University

December 1991

Cooperative Agreement NCC 9-16

Research Activity No. AI.02:
A Computer Graphics Testbed to Simulate and Test Vision System for Space Applications

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

TECHNICAL REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

Summary Report:
A Preliminary Investigation into the Use of Fuzzy Logic for the Control of Redundant Manipulators

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. John B. Cheatham Jr. and Kevin N. Magee of Rice University. Dr. Terry Feagin served as RICIS research coordinator.

Funding was provided by the Information Systems Directorate, Information Technology Division, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this research activity was Dr. Timothy F. Cleghorn of the Information Technology Division, NASA/JSC.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of UHCL, RICIS, NASA or the United States Government.

Summary Report: A Preliminary Investigation into the Use of Fuzzy Logic for the Control of Redundant Manipulators

Research Institute for Computing and Information Systems

Research Activity AI.2

Project: A Computer Graphics Testbed to Simulate and Test Vision Systems for Space Applications

John B. Cheatham, Jr.

Kevin N. Magee

Department of Mechanical Engineering and Materials Sciences

Rice University

December 1991

Summary Report: A Preliminary Investigation into the Use of Fuzzy Logic for the Control of Redundant Manipulators

Abstract

The Rice University Department of Mechanical Engineering and Materials Sciences Robotics Group has designed and built an eight degree of freedom redundant manipulator. Fuzzy logic has been proposed as a control scheme for tasks not directly controlled by a human operator. In preliminary work, fuzzy logic control has been implemented for a camera tracking system and a six degree of freedom manipulator. Both preliminary systems use real-time vision data as input to fuzzy controllers. Related projects include integration of tactile sensing, and fuzzy control of a redundant snakelike arm (under construction).

Outline

1. The ROMR redundant manipulator
   - 1.1 Current control strategy
   - 1.2 Graphic simulation of the ROMR manipulator
2. Preliminary research in fuzzy control
   - 2.1 Fuzzy logic camera tracking system
   - 2.2 Fuzzy control of a six degree of freedom manipulator
3. Future projects
   - 3.1 Integration of tactile sensing
   - 3.2 Fuzzy control in joint coordinates
   - 3.3 Fuzzy control of the ROMR manipulator
   - 3.4 Fuzzy control of a snake-like arm

1. The ROMR redundant manipulator

Although the kinematics and dynamics for redundant robots are difficult to implement, redundant manipulators show remarkable dexterity when compared to their non-redundant counterparts. For example, to make a robot manipulator pass through a narrow hole to retrieve an object, the manipulator must have not only a specific wrist orientation, but also a specific arm configuration. Any similar task requiring simultaneous orientation and configuration specification can not always be accomplished by a non-redundant manipulator, demonstrating the advantages in implementing a redundant structure. [1]

The Rice Omnidirectional Mobile Robot (ROMR) is currently equipped with a single eight degree of freedom redundant manipulator. Since the ROMR is a remote worker in a telepresence system, the ROMR manipulator was constructed to perform complicated manipulation tasks. For this reason the ROMR arm possesses four degrees of freedom for position and four degrees of freedom for orientation. Thus, position and orientation each claim one redundancy. Design specifications and kinematic equations are presented in Appendix A.

1.1 Current control strategy

The control strategy currently employed with the ROMR arm reduces the complexity of the kinematic and dynamic equations by treating the manipulator as a non-redundant robot with variable link parameters. Since the motion of the redundant joints is manually controlled, the automated portion of the control is greatly simplified; however, the burden of choosing configurations is shifted to a human operator. In

short, by treating joint three in the arm and joint five in the wrist as redundant joints (and by temporarily freezing these angles), a configuration and orientation can be determined through inverse kinematics in accordance with the desired trajectory. The solution will not be unique, but by reconfiguring the redundant joints the human operator can select the best configuration and orientation for the task [1].

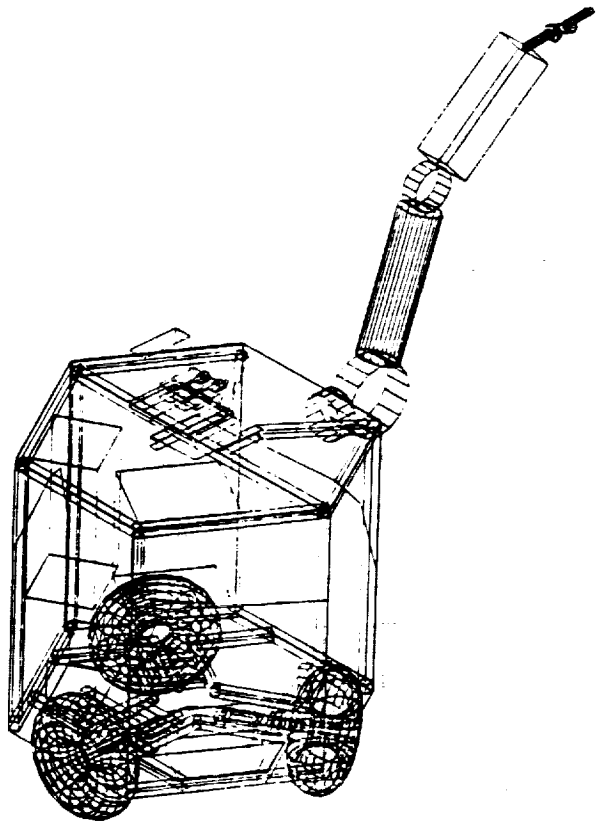


Figure 1: The Rice Omnidirectional Mobile Robot (ROMR) with Eight Degree of Freedom Redundant Manipulator

1.2 Graphics Simulation of the ROMR manipulator

Although the physical system is complete and operational, research with the ROMR manipulator is first implemented in computer graphic

simulation. There are several reasons for developing and experimenting with a robot simulation before implementation on a real robot. First, it is easier and less expensive to make changes to a computer graphics simulation than it is to mechanically or electrically repair a robot. Although successful performance in simulation does not guarantee successful performance in the real world, dangerous and costly failures can be prevented by first repairing them in simulation. Second, if a robot is to be operated either remotely or in hazardous environments, it may be convenient to control the physical robot with respect to a computer simulation rather than with other feedback. This is more easily accomplished if simulation has been incorporated as part of the development process.

The ROMR manipulator simulation has been implemented in the C programming language. Robot motion is controlled by keyboard input, and movement can be specified either in joint coordinates or in world coordinates. Two graphics windows simultaneously show the motion of the manipulator as viewed from two perpendicular viewpoints. The manipulator itself is represented as a simple 2-D line frame, although more detailed 3-D wire frame and solid modeling simulations have been proposed for future work. [1]

2. Preliminary research in fuzzy control

As mentioned earlier, the advantages of kinematic redundancies in robotic manipulators present challenges for control. Although for completely teleoperational tasks the current ROMR manipulator control scheme is sufficient, in many situations it is necessary to provide

robots with local autonomy and control. Because fuzzy logic controllers have proven effective in a variety of other robotics and control applications, fuzzy logic has been proposed as a semi-autonomous control strategy for the ROMR eight degree of freedom redundant manipulator. However, due to the difficulties present in integrating sensor data into control systems, and the complexities inherent in redundant manipulators, there is much technical progress needed before fuzzy control can be applied. Two research projects have been undertaken in preparation for the fuzzy control of the ROMR redundant manipulator.

2.1 Fuzzy logic camera tracking system

To develop a versatile vision guided system, real-time vision data must be integrated into the fuzzy logic controller. As a step towards sensory integration, a fuzzy logic-based camera tracking system has been developed which automatically pans and tilts a camera to keep a selected object centered in the visual field. This is analogous to a robot "turning its head" to follow a moving object.

When the tracking system is started, a live image is displayed on the vision system screen. A human operator then clicks the mouse on an object to be tracked. From then on, the system digitizes the scene and tracks the object at rapid intervals. Currently, the image update/track time is 0.06 seconds, which is equivalent to approximately 17 Hertz. The system continues tracking the object as long as the object remains within the field of view of the camera and its movements are not too

rapid. At every frame update interval a computer simulation of the digitized scene is drawn on the graphics monitor. [2]

2.1.1 The system hardware

The camera tracking system hardware includes a color vision system (TARGA 64 by Truevision) installed on a 486 computer. Modifications were made to an existing camera pan-tilt head to allow it to be controlled from the computer. Additionally, a DC relay control circuit was designed and constructed. The circuit uses three single-pole, double-throw relays to control the direction and speed of each motor. The relays are driven from the computer using a parallel I/O board based around the 8255A PIO chip. Currently the system controls two speeds in both forward and reverse directions, and on/off status for each motor. This requires only six of the twenty four digital ports of the chip, leaving the other I/O lines available for future expansion [2].

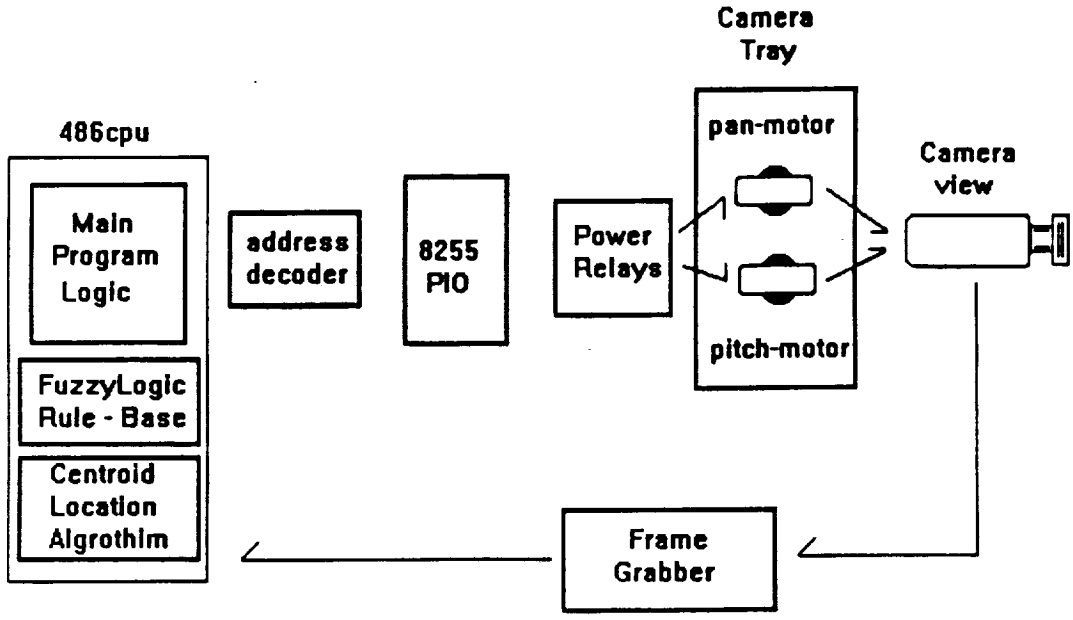


Figure 2: Block diagram of the Fuzzy Logic Camera Tracking System

2.1.2 The fuzzy controller

Through use of a rapid centroid estimation algorithm (presented in Appendix B), the vision processing returns an x and y value in screen coordinates for the location of the object in the visual field. These numbers are fuzzified according to membership functions corresponding to regions of the screen as shown in Figure 3.

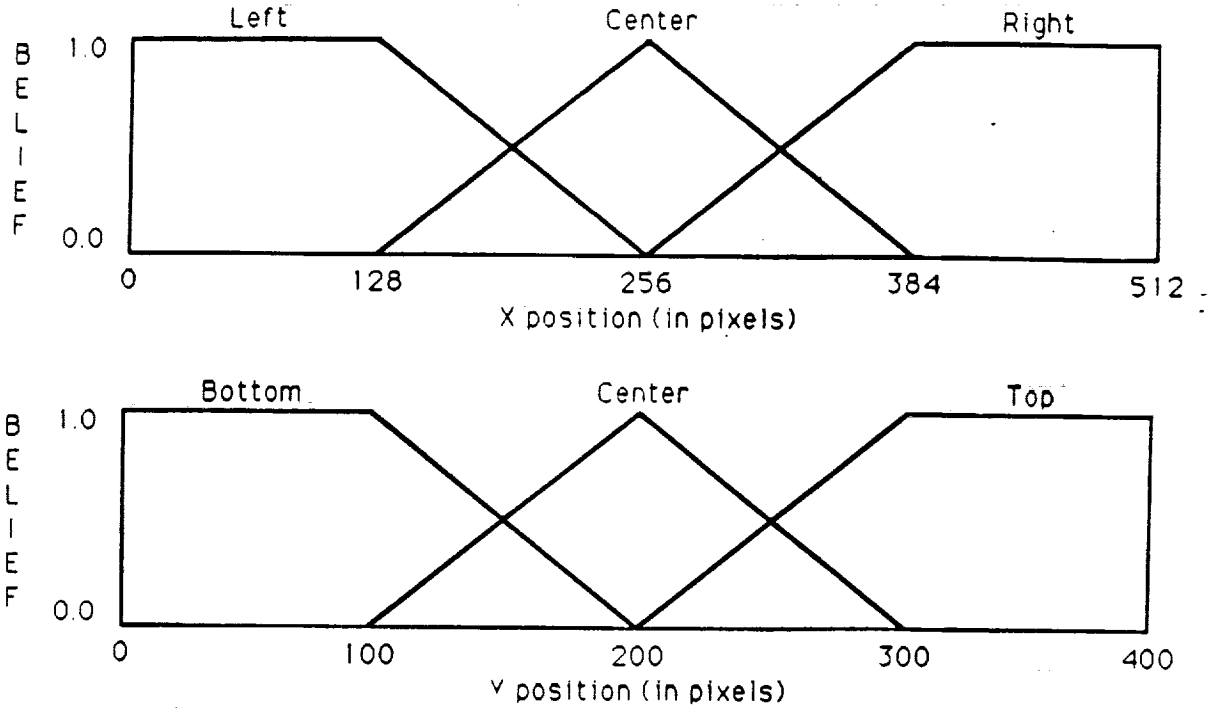


Figure 3: Fuzzy Camera Tracker X and Y Membership Functions

The fuzzy x and y values are used as inputs into the following six rule fuzzy logic rule base, which outputs fuzzy values for the pan and tilt velocities.

```
IF (X IS LEFT) THEN (PAN = LEFT)
IF (X IS CENTER) THEN (PAN = STOP)
IF (X IS RIGHT) THEN (PAN = RIGHT)
IF (Y IS TOP) THEN (TILT = UP)
IF (Y IS CENTER) THEN (TILT = STOP)
IF (Y IS BOTTOM) THEN (TILT = DOWN)
```

The fuzzy pan and tilt velocities are converted to crisp values using centroid defuzzification on the membership functions in Figure 4.

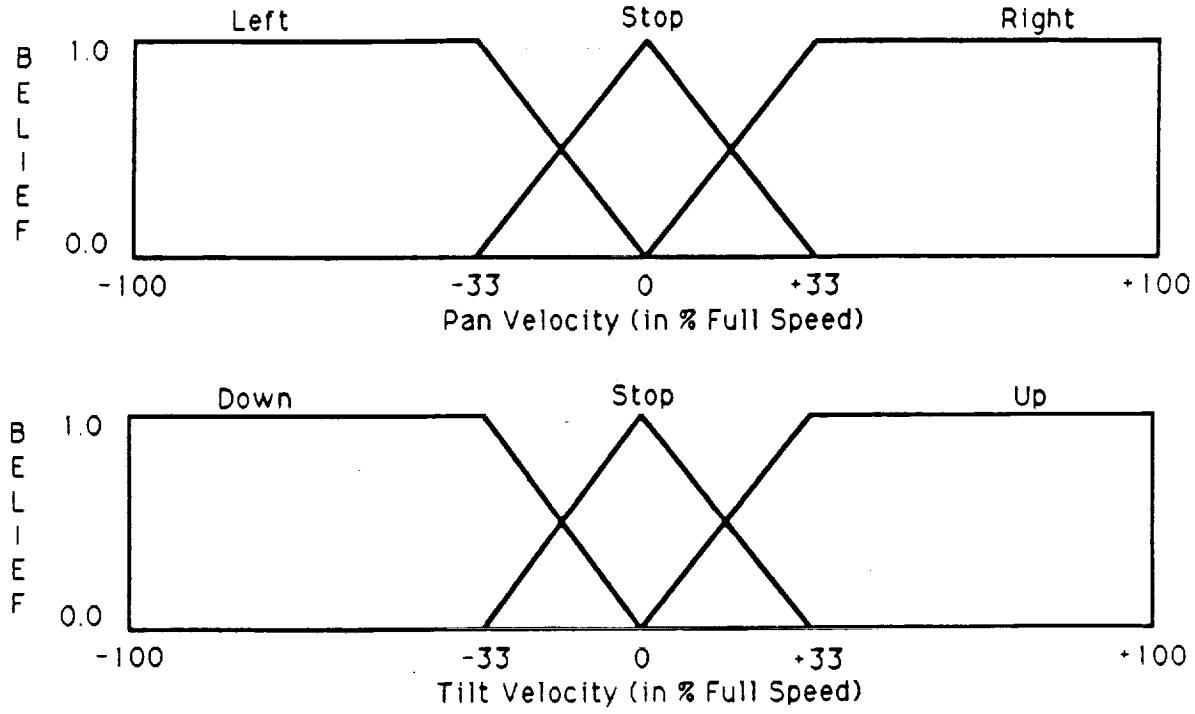


Figure 4: Fuzzy Camera Tracker Pan and Tilt Membership Functions

At this point, the pan and tilt speed values are categorized as either fast, slow, or stop in the indicated directions to accomodate motor

circuitry limitations. Appropriate bits are then set in the I/O board to activate the motors with the desired speeds and directions.

2.2 Fuzzy control of a six degree of freedom manipulator

Building on the integration of visual sensing and fuzzy control accomplished in the fuzzy camera tracker, a fuzzy controller for a six degree of freedom (non-redundant) manipulator has been developed in preparation for the more complicated task of controlling a redundant manipulator. In this system, the user identifies both the robot end effector and the goal object in a live video image by clicking on each of them with the mouse. From then on, the vision software continuously digitizes and interprets the scene while the fuzzy logic rule base specifies motions of the end effector towards the goal until contact is made. The robot is able to locate and grasp either a stationary or a slowly moving object even if the camera is moved during operation. Several innovations over the fuzzy camera tracker include the tracking of multiple objects by the vision system (both a goal object and the robot manipulator instead of only a goal object), and the managing of a higher degree of freedom system by the fuzzy controller (a six degree of freedom robot instead of the two degree of freedom pan-tilt base.)

2.2.1 System hardware

The six degree of freedom manipulator used in this system is a Unimate PUMA 560 industrial robot. A 486 computer equipped with a color vision board performs the image processing and fuzzy logic inferencing. Displacements in world coordinates are generated by the

fuzzy controller and are sent to the PUMA control hardware, which performs the necessary inverse kinematics to shift the robot. The system is arranged as shown in Figure 5.

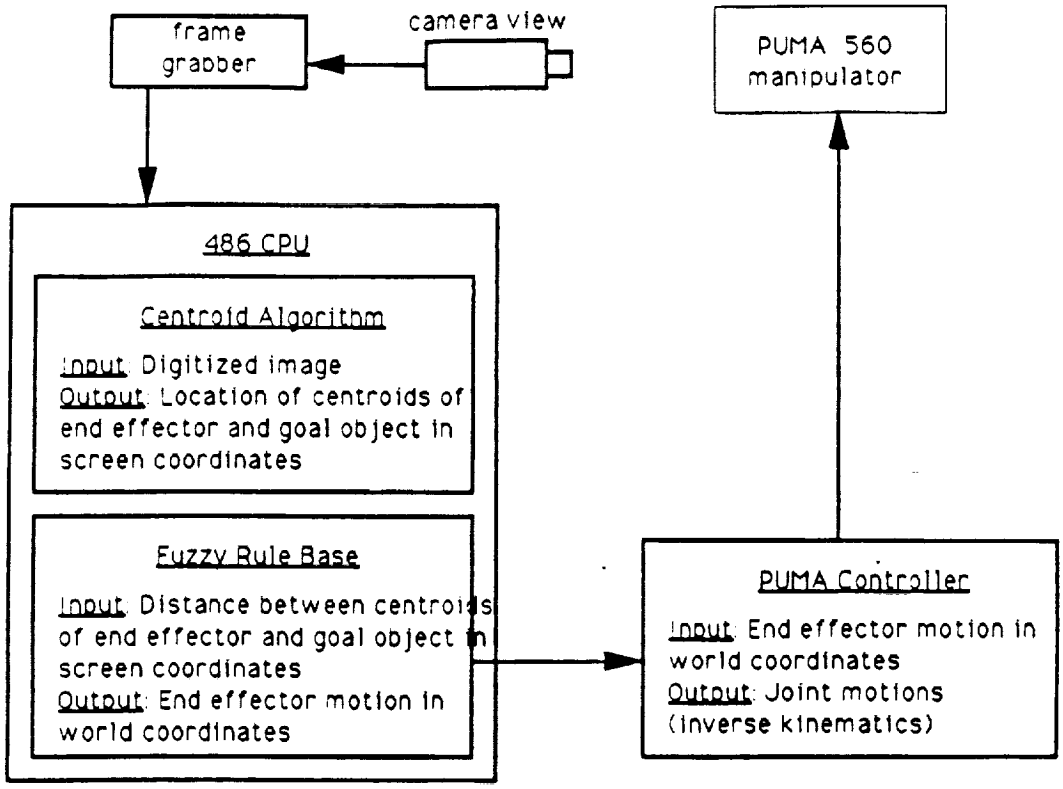


Figure 5: Block diagram of Six Degree of Freedom Manipulator System

2.2.2 The fuzzy controller

The fuzzy controller for this project was developed using the Togai InfraLogic (TIL) shell (see Appendix C). Program listings for the system are provided in Appendix D. The task of the fuzzy controller is to perform a camera-to-world coordinate transformation and to specify world motions by the robot that bring the end effector to rest directly on top of the object. The controller consists of nine rules listed below which are applied to the membership functions shown in Figure 6.

```
IF (DX IS P) THEN (VX = N)
IF (DX IS N) THEN (VX = P)
IF (DX IS Z) THEN (VX = Z)
IF (DX IS NM) THEN (VX = PM)
IF (DX IS PM) THEN (VX = NM)
IF (DY IS P) THEN (VY = N)
IF (DY IS Z) THEN (VY = Z)
IF ((DY IS Z) AND (DX IS Z)) THEN (VY = Z)
IF ((DY IS Z) AND (DX IS NOT Z) THEN (VY = P)
```

Although extension to three spatial dimensions is under development, the current rule base operates in only two dimensions. The specified task is to move the end effector vertically downward onto an object with minimum horizontal motion. Since the object is usually resting on a surface, negative vertical displacements below the object occur infrequently and are treated as zero when they do occur. This results in an asymmetric membership function for y displacement.

One advantage of the fuzzy rule base is that it moves the end effector to a feasible grasp configuration without the specification of intermediate points above the object. In traditional robot programming, an end effector is first moved to a known position precisely above an object and then moved directly downward. The fuzzy controller skips the intermediate step in the following way: At most points in the workspace the end effector is driven towards the goal. However, if the vertical displacement of the end effector from the goal approaches zero while the horizontal displacement is non-zero, the end effector is sent upwards away from the goal. This prevents the end effector from making a final descent until it is appropriately positioned for grasping. The narrow region for zero displacement in the x displacement membership function ensures that a close tolerance is met. Thus, the manipulator grasps the goal object without calculating or traversing a specific intermediate point.

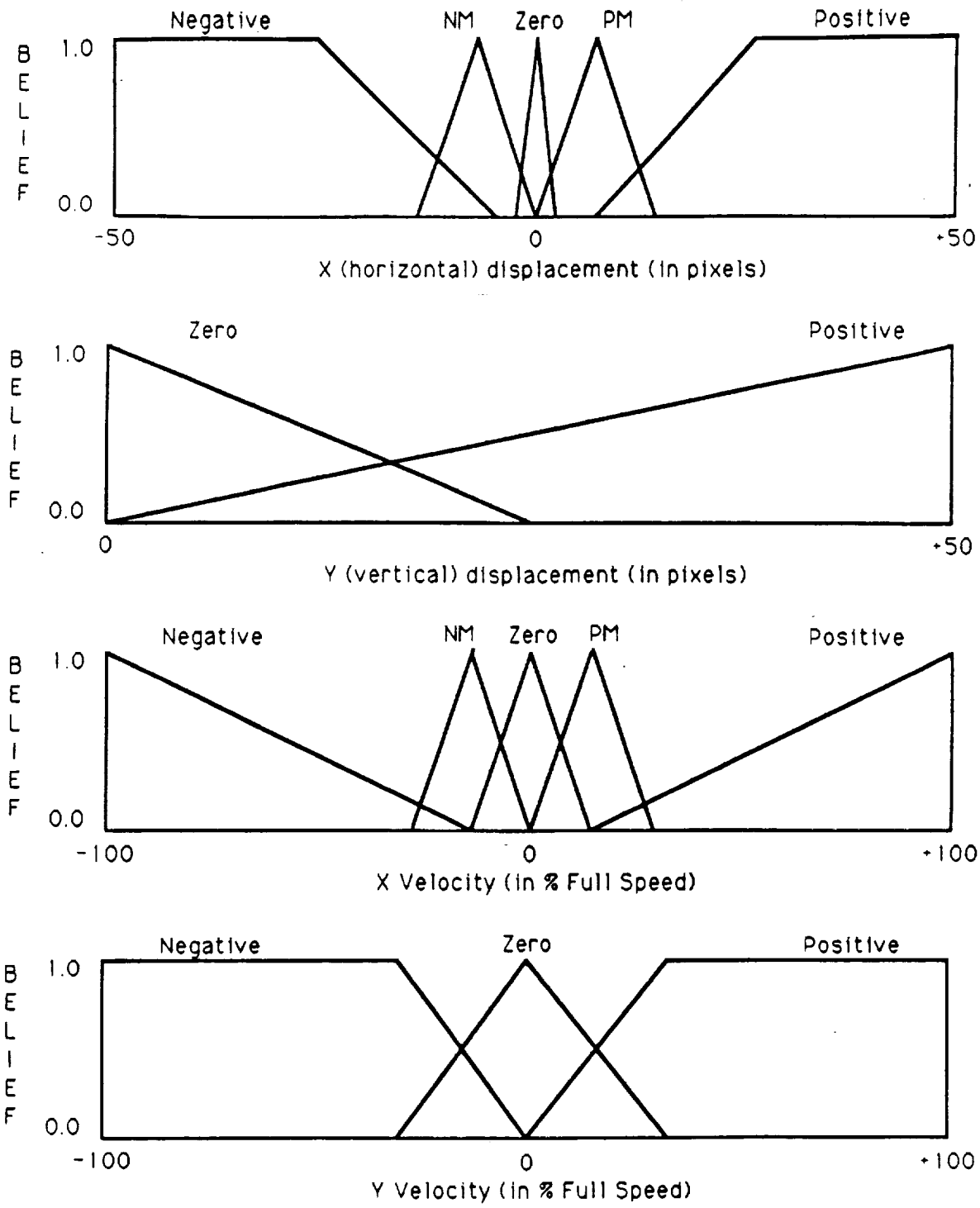


Figure 6: Membership Functions for the Six Degree of Freedom Manipulator Fuzzy Controller

Another advantage of the fuzzy system is that the fuzzy controller needs only to give a rough approximation for the image-to-world coordinate transformation. In the traditional robotic vision approach, a camera must be rigidly fixed and the coordinate transformation between camera and robot must be precisely known. Using fuzzy control, so long as the vision and robot axes are aligned within about twenty degrees, the controller produces displacement values that tend to shift the end effector towards the goal. The only effect of having the axes out of alignment is to alter the velocity specifed by the controller. However, if the misalignment is too severe, the controller will specify a zero velocity too early and the robot will stop without reaching the target. Overall, the robot reaches the destination with great reliability over a wide range of approximate vision and robot axes alignment, though no guaranteed convergence criteria have been developed. This process has proven successful even when the camera is moved during the procedure. In general, the robot is able to track and contact a slowly moving object in real-time, even in the presence of uncertain and changing visual information.

3. Future projects

The end focus of these preliminary research projects has been the effective control of redundant manipulators using fuzzy logic integrated with vision systems. However, several further research steps in sensor integration and fuzzy control are desirable before extension to redundant robots.

3.1 Integration of tactile sensing

Although not essential, it will be useful to integrate tactile sensing into the control scheme to assist in grasping and manipulation tasks. The Robotics Group has constructed two multifingered hands and is carrying out research projects in dextrous manipulation using force and tactile feedback. Using multiple senses will allow manipulation in cases where vision data becomes unavailable, such as when the manipulator blocks its own line of sight. Additionally, force sensing is a proven technique for fine-tuning end effector positions when high precision is required, as it is in many insertion tasks.

3.2 Fuzzy control in joint coordinates

Before applying fuzzy control to redundant manipulators, the fuzzy controller will be modified to transfer its operation from world coordinates to robot joint coordinates. Currently, the fuzzy controller performs an approximate image-coordinate-to-world-coordinate transformation and outputs displacements which bring the end effector closer to the goal object. However, since the fuzzy controller outputs displacements in world coordinates, separate inverse kinematic computations must be performed to appropriately shift the robot joints. A better approach would be to have the fuzzy controller output the necessary joint displacements directly; i.e. to perform approximate inverse kinematics. Because trial and error formulation of the relationships may prove difficult, it is possible that artificial neural networks will be employed to create the fuzzy system. Recent successes have indicated the suitability of neural networks for learning

relationships may prove difficult, it is possible that artificial neural networks will be employed to create the fuzzy system. Recent successes have indicated the suitability of neural networks for learning inverse kinematics [3]. Regardless of how the fuzzy representation is developed, there will be numerous advantages obtained by having the fuzzy controller perform approximate inverse kinematics as well as approximate coordinate camera-to-world coordinate transformations.

A first advantage is that there will be less time spent in kinematic computation. Fuzzy logic inferencing is extremely fast in comparison to calculating traditional inverse kinematics. Additionally, although not yet installed for the six degree of freedom manipulator project, the lab has hardware (the FCD10SB developed by Togai InfraLogic, Inc.) capable of increasing the speed of fuzzy computation by thousands of fuzzy inferences per second.

A second advantage is that no explicit kinematic knowledge of the robot will be required once the appropriate fuzzy rule base and membership functions are developed. Instead, all kinematic formulas will be stored implicitly in the fuzzy rule base. Moreover, no explicit global coordinate system will be employed. After the vision processing has supplied values to the fuzzy controller, the fuzzy controller will specify joint motions corresponding to directions towards the goal, rather than along arbitrary x, y, and z axes.

Although as yet untested, a third advantage may be improved dynamic behavior of the robot. It has been demonstrated that approximate reasoning makes possible the use of an imprecisely positioned camera. In the same way, approximate reasoning on the

A fourth advantage is that it would be possible to include fuzzy optimization as part of the inferencing. To fully exploit the advantages of a redundant arm, it is often desirable to induce self-motion in the robot in such a way as to limit joint velocities, avoid obstacles, and maximize manipulability. Performance measures for these and other criteria have been developed; recent research [4] has used fuzzy logic as a means of combining and applying the various optimization metrics simultaneously. Through fuzzy optimization, important criteria affecting manipulator configuration might be built directly into the controller.

3.3 Fuzzy control of the ROMR manipulator

Once an enhanced fuzzy logic controller has been developed for the six degree of freedom system, construction of a fuzzy logic controller for the eight degree of freedom manipulator will be possible.

The first step will be to develop a controller that works with the manipulator simulation. As with all fuzzy logic development, the bulk of the time will be spent developing rules and membership functions that represent the manipulator, although time may be greatly reduced by using neural networks as mentioned earlier. After the controller has been sufficiently debugged in simulation, it will be applied to the physical ROMR arm. Various manipulation and grasping tasks using vision and any other integrated senses will be performed to evaluate the controller performance.

3.2 Fuzzy control of a snake-like arm

Once control has been accomplished of the ROMR eight degree of freedom manipulator, an attempt will be made to apply fuzzy control to an even more complicated and unusual system.

A team of senior mechanical engineering students, in conjunction with faculty and graduate students of the robotics group, is constructing an eight degree of freedom snake-like manipulator. The design consists of sixteen vertebrae elements, arranged as four sets of four vertebrae each, with each set being capable of 90 degrees of motion in two degrees of freedom. Design predictions indicate that this manipulator will be capable of maneuvering through and around obstacles in much the same manner as a snake [5]. Assuming that a fuzzy logic is successful with the ROMR arm, the snake-like arm presents control problems which seem well suited to the application of fuzzy logic.

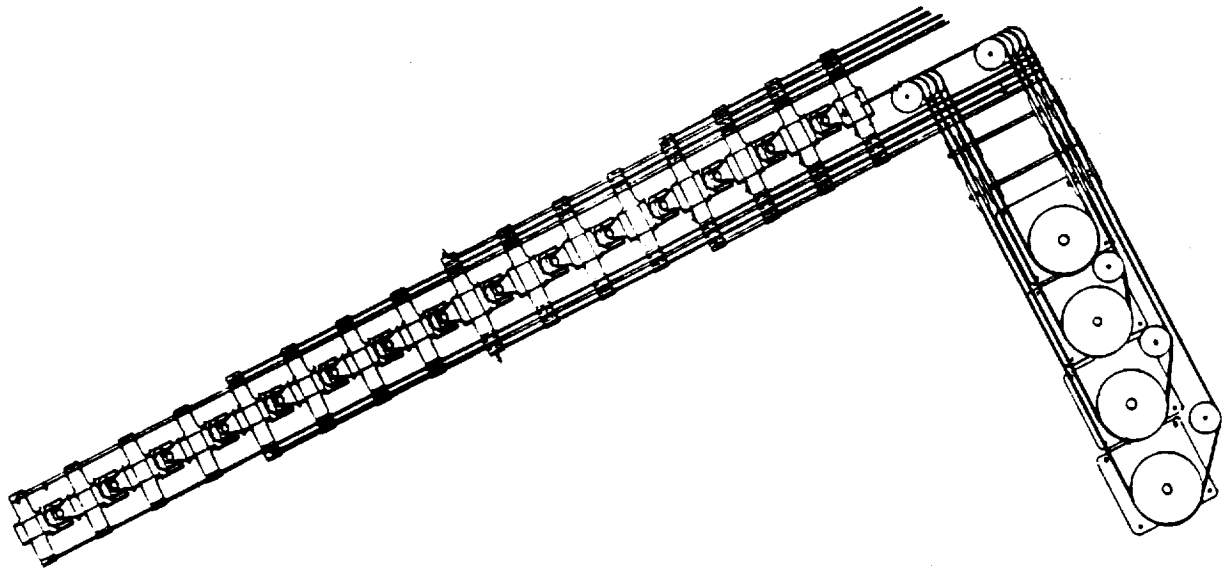


Figure 7: An Eight Degree of Freedom Snake-Like Manipulator

Summary

A description of a currently operational eight degree of freedom redundant manipulator has been presented along with preliminary work in fuzzy control. Success integrating fuzzy control with vision data for the control of a non-redundant manipulator has been described. Research is now focused on extending fuzzy control to the ROMR eight-degree of freedom arm and eventual application to a new snake-like manipulator.

References

[1] Shuxin Gu: "Inverse Kinematics and Computer Simulation of the ROMR Robot", Research Report, Fall 1991.

[2] Sarmad Adnan, "Report on Summer Research in the Robotics Laboratory," Rice University Department of Mechanical Engineering and Materials Sciences, August 16, 1991.

[3] John Norwood, "A Neural Network Approach to the Redundant Robot Inverse Kinematic Problem in the Presence of Obstacles," Doctoral Thesis, Rice University Department of Mechanical Engineering and Materials Sciences, 1990.

[4] Arati Deo, "Redundancy Resolution by Fuzzy Optimization," Project Report for MECH/ELEC698, Rice University, April 1991.

[5] "Design of a Snake-Like Robotic Arm", Rice University, MECH 407 Semester Report, December 4, 1991.

Appendix A

Kinematics of the ROMR robot (taken from [1])

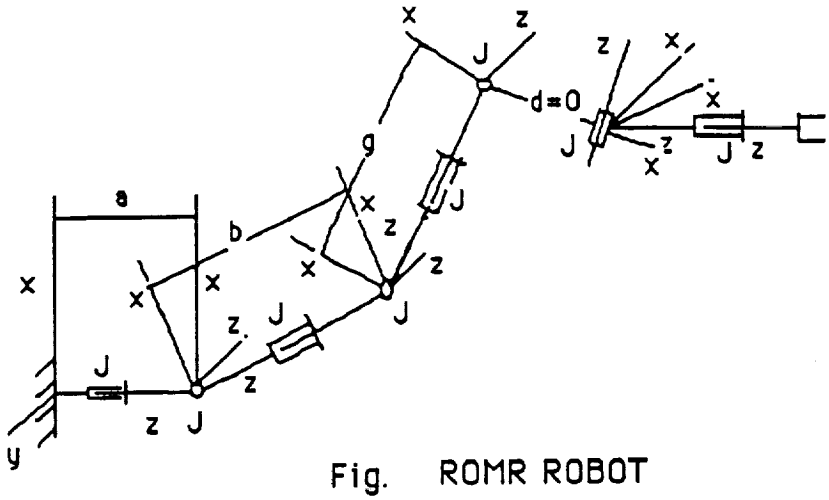


Fig. ROMR ROBOT

D-H CONVENTION TABLE

Link	a_i	α_i	di	θi
1	0	90	a	θ_1
2	0	-90	0	θ_2
3	0	90	b	θ_3
4	0	-90	0	θ_4
5	0	90	g	θ_5
6	0	90	0	θ_6
7	0	90	0	θ_7
8	0	0	0	θ_8

Solution for the all joint angles of ROMR robot:

a) The joint angles in the arm:

θ_3: given

$\theta_4 = \text{atan}(D_2/D_1)$

where:

$$D_1 = (d_x^2+d_y^2+(d_z-a)^2-g^2-b^2)/(2gb)$$
$$D_2 = +(1-\cos(\theta_4)^2)^{1/2} = +(1-D_1^2)^{1/2}$$

$\theta_1 = \text{atan}(E_1/E_2)+\text{atan}(d_y/d_x)$
where:

$$E_1 = g\sin(\theta_3)\sin(\theta_4)/(d_x^2+d_y^2)^{1/2}$$
$$E_2 = +(1-E_1^2)^{1/2}$$

$\theta_2 = \text{atan}((t_4t_1-t_3t_2)/(t_4t_2+t_3t_1))$
where:

$t_1 = -g\cos(\theta_3)\sin(\theta_4)$ $\quad$ $t_2=g\cos(\theta_4)+b$
$t_3 = d_x\cos(\theta_1)+d_y\sin(\theta_1)$ $\quad$ $t_4=d_z-a$

b) The joint angles in the wrist:

θ_5: given

$\theta_6 = \text{atan}(a_y'/a_x')$

$\theta_7 = \text{atan}((-a_x'\cos(\theta_6)-a_y'\sin(\theta_6))/a_z')$

$\theta_8=\text{atan}(F_1/F_2)$
where:

$$F_1 = -(s_x'\cos(\theta_6)+s_y'\sin(\theta_6))\cos(\theta_7)-s_z'\sin(\theta_7)$$
$$F_2 = s_x'\sin(\theta_6)-s_y'\cos(\theta_6)$$

Where:

$$A_0^8 = \begin{bmatrix} n_x & s_x & a_x & d_x \\ n_y' & s_y & a_y & d_y \\ n_z & s_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_6^8 = \begin{bmatrix} n_x' & s_x' & a_x' & 0 \\ n_y' & s_y' & a_y' & 0 \\ n_z' & s_z' & a_z' & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Appendix B

Rapid object tracking software

The object tracking software was developed by Sarmad Adnan to track a single Coke-Bottle-Red object. It was extended by Kevin Magee to track multiple objects with colors determined at run-time. The basic algorithm is as follows:

(1) Digitize an image
(2) Select a pixel on the goal object to be a starting point.
(3) Sequentially check neighboring pixels along the vision system coordinate axes until encountering a pixel with a different color value.
(4) Average the values for the maximum and minimum coordinates along both the x and y axes to obtain a centroid estimate.
(5) To track a moving object, digitize a new image. Loop back to step (3), using the centroid estimate from step (4) as the new starting point.

This fast and simple algorithm works well tracking convex shapes of uniform color. Irregularly-shaped objects and objects with large color variation are tracked less reliably. As with any vision system, performance also depends on the speed with which images can be digitized.

A tracking function using this algorithm appears on page five of the program "viscon.c" (included in appendix D).

Appendix C

Introduction to the Togai InfraLogic Shell

The Togai InfraLogic (TIL) shell is a fuzzy logic development tool created by Togai InfraLogic, Inc. Running under MicroSoft Windows, it provides a graphical interface for creating fuzzy logic systems. The TIL shell allows a user to create high-level objects such as rules, variables, and membership functions. Relationships between objects can also be specified.

Once a fuzzy system has been designed, the user can transform the high level objects into portable C code through the use of a fuzzy C compiler. Various inferencing options and variable representations can be selected at compile time. The C code implementing the fuzzy logic system is readily incorporated into other C programs. Additional options allow a user to produce code that runs on special fuzzy hardware.

Use of the TIL shell in Rice Mechanical Engineering and Materials Sciences robotics lab has greatly reduced the development time of new fuzzy systems. Appendix D includes an example of code produced by the TIL shell.

Appendix D

Control software

The following programs accomplish fuzzy control of the PUMA manipulator using vision sensing; similar programs control the fuzzy logic camera tracking system. The program "world.c", generated by the TIL shell, contains the actual fuzzy logic controller for the manipulator. The program "viscon.c" links together the fuzzy controller with real-time vision data. The program "vision.val", written in the VAL II programming language, receives commands from the fuzzy controller and implements them on the PUMA robot.

```
/*
   HEADER FILE FOR THE VISION DIRECTED FUZZY CONTROLLER FOR ROBOT
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <gf.h>
#include <asiports.h>
#include <targraf.h>
#include <math.h>

/* Communications constants */
#define RXLEN           5
#define TXLEN           100

/* PUMA constants */
   /* Initial position */
#define XINIT           210
#define YINIT           500
#define ZINIT           350
   /* Workspace limits */
#define MAXX            450
#define MAXY            501
#define MAXZ            350
#define MINX              0
#define MINY            499
#define MINZ            -100

/* Vision constants */
#define         COLOR_ERR       0x06L
#define         COLOR_MAX       0x1fL
#define         RED             0x7c00L
#define         GREEN           0x3E0L
#define         BLUE            0x1FL
#define         X_RES           512
#define         Y_RES           400

/* Tracker constants */
#define         MAX             0
#define         MIN             1
#define         NOB             2
#define         ROBOT           0
#define         GOAL            1

/* General definitions */
#define         BETWEEN(x,y,z)  ((x)>=(y)) && ((x)<=(z))
#define         FALSE           0
#define         TRUE            1

extern int ok;

/* Vision functions */
void vision_init(void);
unsigned long int locate_object(int *blob_x, int *blob_y);
void track_object(int objx[NOB], int objy[NOB], int *track);
void define_color_range(int object, unsigned long int color);
void draw_cross(int x, int y);

\032
```

```
/*
   VISION DIRECTED FUZZY CONTROLLER FOR ROBOT

   This version has only one way communication from the fuzzy controller
   to the robot.  All feedback on robot position is entirely from vision.

   For better synchronization between the robot and fuzzy controller, it
   is helpful to do the following:
      (1) Have the robot move at high speed
      (2) Specify only small displacements in each command

   The fuzzy controller, developed with the TIL shell, is programmed in
   a separate file.

*/

#include "viscon.h"
#include <math.h>

#include <tilcomp.h>
#include "world.h"

#define VISLOOP         1
#define CONTACT         50.0
#define HORZERR         200.0
#define INRANGE         10.0
#define TIME            2

/* Global color ranges */
unsigned long int  red[NOB][2], green[NOB][2], blue[NOB][2];

main()
{
   int             x[NOB], y[NOB];
   int             track;
/*   int             sim_color[NOB]={5,4}; */
   unsigned long   color;
   unsigned int    button;
   int             px=210, py=500, pz=325, ptx, ptz;
   int             dpx=0, dpy=0, dpz=0, chk=0;
   char            s[TXLEN+1];
   double          xd, yd, dist;
   int             xdist, ydist;

   /* Initialize communications with PUMA on COM2 */
   printf("Initializing communications...\n");
   asifirst(COM2, (ASINOUT|BINARY|NORMALRX), RXLEN, TXLEN);
   printf("Step two in comm init.\n");
   asiinit(COM2, 9600, P_NONE, 1, 8);

   /* Initialize the vision system */
   printf("Initializing vision...\n");
   vision_init();

   /* Start up the slave program on the PUMA */
   sprintf(s,"ex vision%c", 10);
   printf("Starting PUMA:    %s", s);
   asputs(COM2, s, 13);

   /* Make sure user knows what is going on */
   printf("\nMake certain that both robot and target ");
   printf("are stationary and in view.\n");

   /* Wait for user to release mouse button */
```

```
   button = TRUE;
   while( button != FALSE)
      Buttons(&button);

   /* Locate the robot and the goal */
   printf("\nClick on the robot\n");
   color = locate_object( &x[ROBOT], &y[ROBOT]);
   define_color_range(ROBOT, color);
   printf("Robot color = %x\n", color);

   printf("\nClick on the goal.\n");
   color = locate_object( &x[GOAL], &y[GOAL]);
   define_color_range(GOAL, color);
   printf("Goal color = %x\n", color);

   do {
      /* Track the objects */
      track_object( x, y, &track);

      /* How far apart are the centroids in vision coords? */
      xdist = (x[GOAL] - x[ROBOT]);
      ydist = (y[ROBOT] - y[GOAL]);
      xd = (double) xdist;
      yd = (double) ydist;
      dist = sqrt(xd*xd + yd*yd);

      /* Only move the robot if you are tracking! */
      if (track == TRUE) {

         /* Move if the last specified motion was non-zero */
         if (dist > CONTACT) {
            /* Send the final position values of the E.E. and G. to the
               Fuzzy Controller.  It will output displacement values
               for the robot to move the end effector closer to the goal. */
            FuzzyWorldControl(xdist, ydist,  &dpx, &dpz);

            /* Convert the output velocity into a displacement by multiplying
               by the time factor (actually divide by 1/factor) */
            dpx /= TIME;
            dpz /= TIME;

            /* Compute the possible new position of the PUMA robot */
            ptx = px + dpx,  ptz = pz + dpz;

            /* Make sure values are in workspace window of PUMA */
            if ((ptx<MAXX) && (ptx>MINX)) px = ptx;
            if ((ptz<MAXZ) && (ptz>MINZ)) pz = ptz;

            /* Send the displacement values and a checksum */
            chk = dpx + dpz;
            sprintf(s,"%d %d %d %d%c", dpx, 0, dpz, chk, 10);
            /* printf("Sending displacements:    %s", s); */
            asputs(COM2, s, 13);
         }
         else { /* IF contact has been made! */
            printf("In position!  Hit a key to continue...\n");
            sprintf(s,"%d %d %d %d%c", 0, 10, 0, 0, 10);
            asputs(COM2, s, 13);

            /* Return PUMA to ready position */
            sprintf(s,"%d %d %d %d%c", 0, 255, 0, 0, 10);
            asputs(COM2, s, 13);

            while(!kbhit());
```

```
         }
      }
      else {  /* IF _NOT_ TRACKING */
         printf("%c",7);
      }

   } while (!kbhit());

   /* Send code to the PUMA to make it halt */
   sprintf(s,"%d %d %d %d%c", 0, 0, 0, 255, 10);
   printf("Sending termination code:    %s", s);
   asputs(COM2, s, 13);

   asiquit(COM2);

   exit(0);
}



/* ************************************************************************ */
/* ---------------------- VISION FUNCTIONS ---------------------------------- */
/* ************************************************************************ */



/* Initialize the vision board and mouse */
void vision_init(void)
{
   if (InitGraphics() != 0) {
      printf("Targa board not responding, install device driver\n");
      exit(1);
   }
   if(OpenPntDev(microsoftMouse) == FALSE) {
      printf("Mouse driver not found\n");
      exit(1);
   }
   if (SelectLiveSource(vidInputComposite) == FALSE) {
      fprintf(stderr, "Can not select video source\n");
      exit(1);
   }
   EnableGenlock();
   EnableDisplay();
}

/* Allow a user to choose an object to track (called once for each object)
   An object can either be a goal or the end-effector */
unsigned long int locate_object(int* blob_x, int* blob_y)
{
   unsigned int btn_state;
   unsigned long color, delay;
   Point pt;

   do {
      GrabFrame();
      GetPDevPos(&pt);
      draw_cross(pt.x, pt.y), draw_cross(pt.x+4, pt.y), draw_cross(pt.x-4, pt.y), draw_cro
ss(pt.x, pt.y+4), draw_cross(pt.x, pt.y-4);
      Buttons(&btn_state);
      if(kbhit()) {
         printf("two: keyboard touched, exiting\n");
         exit(0);
      }
```

```
      draw_cross(pt.x, pt.y), draw_cross(pt.x+4, pt.y), draw_cross(pt.x-4, pt.y), draw_cr
ss(pt.x, pt.y+4), draw_cross(pt.x, pt.y-4);
      for(delay=0; delay<10000; delay++);
   } while( !(btn_state & leftButton) );

   /* Wait for user to release button */
   while( (btn_state & leftButton) )
      Buttons(&btn_state);

   /* Find out the color of the selected object */
   GetPixel(pt.x, pt.y, &color);

   *blob_x=pt.x;
   *blob_y=pt.y;
   return color;
}



/* Create a range of color to around the selected object color since
   objects are never of truly uniform color.  Good color range values
   are chosen through experimentation. */
void define_color_range(int object, unsigned long int color)
{
    unsigned long int redval, greenval, blueval;

    redval = (color & RED)>>10;
    red[object][MIN] = ((redval<COLOR_ERR)?(0L):(redval-COLOR_ERR))<<10;
    red[object][MAX] = ((redval+COLOR_ERR>COLOR_MAX)?(COLOR_MAX):(redval+COLOR_ERR))<<10;
    printf("red:  min = %lx    max = %lx\n", red[object][MIN], red[object][MAX]);

    greenval = (color & GREEN)>>5;
    green[object][MIN] = ((greenval<COLOR_ERR)?(0L):(greenval-COLOR_ERR))<<5;
    green[object][MAX] = ((greenval+COLOR_ERR>COLOR_MAX)?(COLOR_MAX):(greenval+COLOR_ERR)
<<5;
    printf("green:  min = %lx    max = %lx\n", green[object][MIN], green[object][MAX]);

    blueval = (color & BLUE);
    blue[object][MIN] = ((blueval<COLOR_ERR) ? (0L) : (blueval-COLOR_ERR));
    blue[object][MAX] = ((blueval+COLOR_ERR>COLOR_MAX)?(COLOR_MAX):(blueval+COLOR_ERR));
    printf("blue:  min = %lx    max = %lx\n", blue[object][MIN], blue[object][MAX]);

}


/* Draw the cursor on the vision screen for mouse usage and for locating
   lost objects. */
void draw_cross(int x, int y)
{
   unsigned long xc = RED;

   SetPixel(x-2, y, xc), SetPixel(x-1, y, xc), SetPixel(x+1, y, xc), SetPixel(x-1, y, xc)
 SetPixel(x, y-2, xc), SetPixel(x, y-1, xc), SetPixel(x, y+1, xc), SetPixel(x, y+2, xc);
}


/* The fast centroid estimation algorithm.  Performed once for each object
   (either a goal or an end effector) */
void track_object(int objx[NOB], int objy[NOB], int *track)
{
   unsigned long  color, redmin, redmax, greenmin, greenmax, bluemin, bluemax;
   register int   top, bottom, right, left;
   int x, y, object;
```

```
   GrabFrame();
   *track = TRUE;

   for(object=0; object<NOB; object++) {
      redmin   =  red[object][MIN];
      redmax   =  red[object][MAX];
      greenmin =  green[object][MIN];
      greenmax =  green[object][MAX];
      bluemin  =  blue[object][MIN];
      bluemax  =  blue[object][MAX];

      x = objx[object];
      y = objy[object];

      GetPixel(x, y, &color);

      if ( BETWEEN(color&RED,redmin,redmax) && BETWEEN(color&GREEN,greenmin,greenmax) && B
ETWEEN(color&BLUE,bluemin,bluemax) )  {
         /* Search up until finding top of object */
         for(top=y; top<Y_RES; top++) {
            GetPixel(x, top, &color);
            if( !(BETWEEN(color&RED,redmin,redmax) && BETWEEN(color&GREEN,greenmin,greenma
x) && BETWEEN(color&BLUE,bluemin,bluemax) ) )
               break;
         }

         /* Search down until finding bottom of object */
         for(bottom=y; bottom>=0; bottom--) {
            GetPixel(x, bottom, &color);
            if( !(BETWEEN(color&RED,redmin,redmax) && BETWEEN(color&GREEN,greenmin,greenma
x) && BETWEEN(color&BLUE,bluemin,bluemax) ) )
               break;
         }

         /* Search to the right until finding the right edge */
         for(right=x; right<X_RES; right++) {
            GetPixel(right, y, &color);
            if( !(BETWEEN(color&RED,redmin,redmax) && BETWEEN(color&GREEN,greenmin,greenma
x) && BETWEEN(color&BLUE,bluemin,bluemax) ) )
               break;
         }

         /* Search to the left until finding the left edge */
         for(left=x; left>=0; left--) {
            GetPixel(left, y, &color);
            if( !(BETWEEN(color&RED,redmin,redmax) && BETWEEN(color&GREEN,greenmin,greenma
x) && BETWEEN(color&BLUE,bluemin,bluemax) ) )
               break;
         }

         /* Calculate the centroid estimate */
         objx[object] = (right+left)/2;
         objy[object] = (top+bottom)/2;
       }

       else {
         *track = FALSE;
         draw_cross(x, y), draw_cross(x+4, y), draw_cross(x-4, y), draw_cross(x, y+4), dra
w_cross(x, y-4);
       }
   }
}
\032
```

```
/*
 * Togai InfraLogic (R) Fuzzy-C Compiler Version 2.3.1
 *
 * Source file: "world.til"
 *
 * Compiled Fri Dec 06 14:55:05 1991
 *
 * Selected options:
 * Default map size:            256
 * Default map type:            FUBYTE
 * Output C format:             OFF
 * Emit unused MEMBERs:         ON
 * Force expr:                  OFF
 * Force map:                   OFF
 * Generate debug code:         OFF
 */

struct _SWO_b_point { SWORD x; FUBYTE y; };

void FuzzyWorldControl (SWORD X, SWORD Y, SWORD *VX, SWORD *VY);
```

```
/*
 * Togai InfraLogic (R) Fuzzy-C Compiler Version 2.3.1
 *
 * Source file: "world.til"
 *
 * Compiled Fri Dec 06 14:55:05 1991
 *
 * Selected options:
 * Default map size:            256
 * Default map type:            FUBYTE
 * Output C format:             OFF
 * Emit unused MEMBERs:         ON
 * Force expr:                  OFF
 * Force map:                   OFF
 * Generate debug code:         OFF
 */

/*
 * Portability definitions and internal structures.
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <tilcomp.h>
#include "world.h"

#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)<(b)?(b):(a))

struct _til_cbinfo { long moment, area; };
struct _til_cfinfo { double moment, area; };
struct _til_hbinfo { long hcentroid, height; };
struct _til_hfinfo { double hcentroid, height; };

/*
 * Point list evaluation routine for signed word values.
 */

static FUBYTE _SWO_b_ploop (SWORD var, int n, struct _SWO_b_point *pts);

static FUBYTE _SWO_b_ploop (SWORD var, int n, struct _SWO_b_point *pts)
{
    register int i;
    FUBYTE _alpha;
    SLONG _tmp;

    if (var < pts->x)
        _alpha = ((FUBYTE) 0);

    else if (var == pts->x)
        _alpha = pts->y;

    else {
        _alpha = ((FUBYTE) 0);

        for (i = 0; i < n - 1; i++, pts++)
            if (var > pts->x && var <= (pts+1)->x) {
                _tmp =   (((SLONG) (var)) - ((SLONG) pts->x)) *
                         (((SLONG) (pts+1)->y) - ((SLONG) pts->y));
                _alpha = (FUBYTE) (((SLONG) pts->y) + _tmp /
                         (((SLONG) (pts+1)->x) - ((SLONG) pts->x)));
                break;
            }
```

```
    }

    return _alpha;
}

static FUBYTE _VX_N_map[] = {
/*00000*/ 255, 248, 242, 235, 229, 223, 216, 210, 204, 197,
/*00010*/ 191, 184, 178, 172, 165, 159, 153, 146, 140, 133,
/*00020*/ 127, 121, 113, 108, 102,  95,  89,  81,  76,  70,
/*00030*/  63,  56,  51,  44,  38,  31,  25,  19,  12,   6,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0
};

static FUBYTE _VX_Z_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,  65, 160,
/*00050*/ 255, 160,  65,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0
};

static FUBYTE _VX_P_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   6,  12,  19,  25,  31,  38,  44,  51,  56,
/*00070*/  63,  70,  76,  81,  89,  95, 102, 108, 113, 121,
/*00080*/ 127, 133, 140, 146, 153, 159, 165, 172, 178, 184,
/*00090*/ 191, 197, 204, 210, 216, 223, 229, 235, 242, 248,
/*00100*/ 255
};

static FUBYTE _VX_NM_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,  25,  51,  76, 102, 127, 153, 178, 204, 229,
/*00040*/ 255, 229, 204, 178, 153, 127, 102,  76,  51,  25,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0
};

static FUBYTE _VX_PM_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,  25,  51,  76, 102, 127, 153, 178, 204, 229,
/*00060*/ 255, 229, 204, 178, 153, 127, 102,  76,  51,  25,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0
};

static FUBYTE _VY_N_map[] = {
/*00000*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00010*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00020*/ 255, 255, 255, 255, 255, 255, 243, 234, 224, 214,
/*00030*/ 204, 193, 183, 173, 163, 153, 142, 131, 121, 112,
/*00040*/ 102,  91,  81,  71,  60,  51,  40,  30,  20,  10,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0
};

static FUBYTE _VY_Z_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,  10,  20,  30,  40,
/*00030*/  51,  60,  71,  81,  91, 102, 112, 121, 131, 142,
/*00040*/ 153, 163, 173, 183, 193, 204, 214, 224, 234, 243,
/*00050*/ 255, 243, 234, 224, 214, 204, 193, 183, 173, 163,
/*00060*/ 153, 142, 131, 121, 112, 102,  91,  81,  71,  60,
/*00070*/  51,  40,  30,  20,  10,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0
};

static FUBYTE _VY_P_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,  10,  20,  30,  40,  51,  60,  71,  81,  91,
/*00060*/ 102, 112, 121, 131, 142, 153, 163, 173, 183, 193,
/*00070*/ 204, 214, 224, 234, 243, 255, 255, 255, 255, 255,
/*00080*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00090*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00100*/ 255
};

/*
 * User defined objects.
 */

/*
 * VAR X
 */

/*
 * MEMBER N
 */
```

```
static struct _SWO_b_point _X_N_b_pts[] = {
    { -500, 255 },
    { -50, 0 }
};

/*
 * MEMBER Z
 */

static struct _SWO_b_point _X_Z_b_pts[] = {
    { -13, 0 },
    { 0, 255 },
    { 18, 0 }
};

/*
 * MEMBER P
 */

static struct _SWO_b_point _X_P_b_pts[] = {
    { 50, 0 },
    { 500, 255 }
};

/*
 * MEMBER NM
 */

static struct _SWO_b_point _X_NM_b_pts[] = {
    { -100, 0 },
    { -52, 252 },
    { 0, 0 }
};

/*
 * MEMBER PM
 */

static struct _SWO_b_point _X_PM_b_pts[] = {
    { 0, 0 },
    { 50, 255 },
    { 100, 0 }
};

/*
 * VAR Y
 */

/*
 * MEMBER Z
 */

static struct _SWO_b_point _Y_Z_b_pts[] = {
    { 0, 255 },
    { 120, 0 }
};

/*
 * MEMBER P
 */

static struct _SWO_b_point _Y_P_b_pts[] = {
    { 0, 0 },
```

```
    { 250, 255 },
    { 500, 255 }
};

/*
 * VAR VX
 */

/*
 * VAR VY
 */

/*
 * FUZZY Controller
 */

static void Controller (SWORD X, SWORD Y, SWORD *VX, SWORD *VY);

static void Controller (SWORD X, SWORD Y, SWORD *VX, SWORD *VY)
{
    FUBYTE _alpha;
    FUBYTE _X_is_Z;
    FUBYTE _Y_is_Z;
    FUBYTE _VX_alpha;
    FUBYTE _VX_temp[101];
    FUBYTE _VY_alpha;
    FUBYTE _VY_temp[101];
    register int i;
    FLOAT _moment;
    FLOAT _area;

    memset (_VX_temp, 0, sizeof (_VX_temp));
    memset (_VY_temp, 0, sizeof (_VY_temp));

    _X_is_Z = _SWO_b_ploop (X, 3, _X_Z_b_pts);
    _Y_is_Z = _SWO_b_ploop (Y, 2, _Y_Z_b_pts);

/*
 * RULE Rule0000
 */

    _alpha = _SWO_b_ploop (X, 2, _X_P_b_pts);

    if (_alpha != ((FUBYTE) 0)) {

        /* VX = N */

        for (i = 0; i < 101; i++) {
            _VX_alpha = min (_alpha, _VX_N_map[i]);
            if (_VX_temp[i] < _VX_alpha)
                _VX_temp[i] = _VX_alpha;
        }

    }

/*
 * RULE Rule0001
 */

    _alpha = _SWO_b_ploop (X, 2, _X_N_b_pts);

    if (_alpha != ((FUBYTE) 0)) {

        /* VX = P */
```

```
        for (i = 0; i < 101; i++) {
            _VX_alpha = min (_alpha, _VX_P_map[i]);
            if (_VX_temp[i] < _VX_alpha)
                _VX_temp[i] = _VX_alpha;
        }

    }

/*
 * RULE Rule0002
 */

    _alpha = _X_is_Z;

    if (_alpha != ((FUBYTE) 0)) {

        /* VX = Z */

        for (i = 0; i < 101; i++) {
            _VX_alpha = min (_alpha, _VX_Z_map[i]);
            if (_VX_temp[i] < _VX_alpha)
                _VX_temp[i] = _VX_alpha;
        }

    }

/*
 * RULE Rule0006
 */

    _alpha = _SWO_b_ploop (X, 3, _X_NM_b_pts);

    if (_alpha != ((FUBYTE) 0)) {

        /* VX = PM */

        for (i = 0; i < 101; i++) {
            _VX_alpha = min (_alpha, _VX_PM_map[i]);
            if (_VX_temp[i] < _VX_alpha)
                _VX_temp[i] = _VX_alpha;
        }

    }

/*
 * RULE Rule0007
 */

    _alpha = _SWO_b_ploop (X, 3, _X_PM_b_pts);

    if (_alpha != ((FUBYTE) 0)) {

        /* VX = NM */

        for (i = 0; i < 101; i++) {
            _VX_alpha = min (_alpha, _VX_NM_map[i]);
            if (_VX_temp[i] < _VX_alpha)
                _VX_temp[i] = _VX_alpha;
        }

    }

/*
```

```
 * RULE Rule0003
 */

    _alpha = _SWO_b_ploop (Y, 3, _Y_P_b_pts);

    if (_alpha != ((FUBYTE) 0)) {

        /* VY = N */

        for (i = 0; i < 101; i++) {
            _VY_alpha = min (_alpha, _VY_N_map[i]);
            if (_VY_temp[i] < _VY_alpha)
                _VY_temp[i] = _VY_alpha;
        }

    }

/*
 * RULE Rule0004
 */

    _alpha = min(_Y_is_Z, _X_is_Z);

    if (_alpha != ((FUBYTE) 0)) {

        /* VY = Z */

        for (i = 0; i < 101; i++) {
            _VY_alpha = min (_alpha, _VY_Z_map[i]);
            if (_VY_temp[i] < _VY_alpha)
                _VY_temp[i] = _VY_alpha;
        }

    }

/*
 * RULE Rule0005
 */

    _alpha = min(_Y_is_Z, ((FUBYTE) 255) - _X_is_Z);

    if (_alpha != ((FUBYTE) 0)) {

        /* VY = P */

        for (i = 0; i < 101; i++) {
            _VY_alpha = min (_alpha, _VY_P_map[i]);
            if (_VY_temp[i] < _VY_alpha)
                _VY_temp[i] = _VY_alpha;
        }

    }


    _moment = _area = 0.0;
    for (i = 0; i < 101; i++) {
        _area += ((FLOAT) _VX_temp[i]);
        _moment += ((FLOAT) _VX_temp[i]) * (FLOAT) i;
    }
    if (_area != 0.000000)
        *VX = ((SWORD) ((_moment / _area) + -50.0000));

    _moment = _area = 0.0;
    for (i = 0; i < 101; i++) {
```

```
        _area += ((FLOAT) _VY_temp[i]);
        _moment += ((FLOAT) _VY_temp[i]) * (FLOAT) i;
    }
    if (_area != 0.000000)
        *VY = ((SWORD) ((_moment / _area) + -50.0000));

}

/*
 * PROJECT FuzzyWorldControl
 */

void FuzzyWorldControl (SWORD X, SWORD Y, SWORD *VX, SWORD *VY)
{
/*
 * FUZZY Controller
 */

    Controller (X, Y, VX, VY);

}
```

```
EDIT vision
SPEED 50*100/SPEED(1) MMPS ALWAYS
CLOSEI
FLIP
NONULL
COARSE
SET a = TRANS(210, 500, 325, -127, 87, 152)
MOVE a
BREAK
no.error = TRUE
WHILE  no.error DO
PROMPT "", x, g, z, c
IF c == (x+z) THEN
IF g == 0 THEN
SET a = SHIFT(a BY x, y, z)
MOVE a
ELSE
IF g == 255 THEN
READY
ELSE
OPENI
END
END
ELSE
no.error = FALSE
END
END
NULL
FINE
E
```